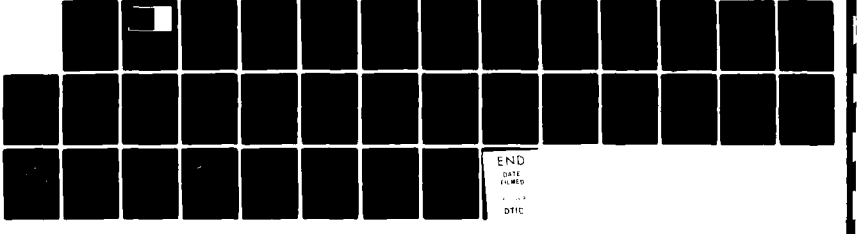
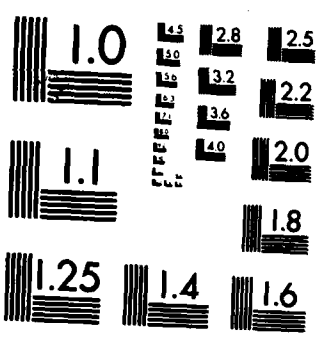
AD-A124 348 HIDDEN LINE ELIMINATION IN PROJECTED GRID SURFACES(U) 1/1
WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER
D P ANDERSON DEC 82 MRC-TSR-2447 DAAG29-80-C-0041

UNCLASSIFIED F/G 12/1 NL





MRC Technical Summary Report #2447

HIDDEN LINE ELIMINATION IN PROJECTED GRID SURFACES

David P. Anderson

Mathematics Research Center
University of Wisconsin—Madison
610 Walnut Street
Madison, Wisconsin 53706

December 1982

(Received November 3, 1982)

Approved for public release
Distribution unlimited

Sponsored by

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

National Science Foundation
Washington, D. C. 20550

UNIVERSITY OF WISCONSIN-MADISON

MATHEMATICS RESEARCH CENTER

HIDDEN LINE ELIMINATION IN PROJECTED GRID SURFACES

David P. Anderson

Technical Summary Report #2447
December 1982

ABSTRACT

Hidden line and hidden surface problems are often simpler when restricted to special classes of objects. An example is the class of grid surfaces, i.e. graphs of bivariate functions represented by their values on a set of grid points. Projected grid surfaces have geometric properties which permit hidden line or hidden surface elimination to be done more easily then in the general case. These properties are discussed in this paper and an algorithm is given which exploits them.

CR Category Number: I.3.7 (Visible line/surface algorithm)
Key Words: Hidden line elimination, hidden surface elimination, function graphing, grid surface
Work Unit Number 5: Mathematical Programming

Sponsored by the United States Army under Contract No. DAAG29-80-C-0041. This material is based on work supported by National Science Foundation Grant MCS-8200632.

SIGNIFICANCE AND EXPLANATION

A problem of great practical importance is that of graphically constructing realistic 2-dimensional models of 3-dimensional grid surfaces. In this report a fast algorithm is developed for doing this. A number of complex surfaces are modelled to illustrate the power of the algorithm.

The responsibility for the wording and views expressed in this descriptive summary lies with MRC, and not with the author of this report.

HIDDEN LINE ELIMINATION IN PROJECTED GRID SURFACES

David P. Anderson

1) INTRODUCTION

The graph of a function defined on a rectangle is a surface in three-space, and can be represented in a two-dimensional medium by a contour map or a projected image. While a contour map contains more exact information about the function, a projected image is more helpful in visualizing the surface's shape. In practice, a function is often presented as a set of values on the points of a grid. Its graph can then be approximated by a "grid surface" consisting of straight-edged regions. The task of generating images of grid surfaces can be performed by general hidden line algorithms (ref. 3, 5). However, the relative slowness of these algorithms has prompted the development of methods specifically for grid surfaces (ref. 2, 4, 6, 7).

Previous methods for drawing grid surfaces have achieved speed at the expense of exactness and generality. Butland's algorithm (ref.2) is exact and linear-time but is restricted to parallel projection using a viewing direction whose projection on the xy plane makes an angle with the x axis which is a multiple of 45 degrees. The algorithm of Kubert, Szabo and Gulieri (ref. 4) uses time of order $n^{1.5}$ and is not exact because segments which are partially hidden are not drawn at all. Williamson's algorithm (ref.6) can be used to draw x- or y-direction lines in the grid exactly, but not both. Furthermore, it is based on the assumption

Sponsored by the United States Army under Contract No. DAAG29-80-C-0041. This material is based on work supported by National Science Foundation Grant MCS-8200632.

that the perimeter of the image of a grid surface can be divided into upper and lower parts such that all "vertical" lines in the image space intersect each part at most once. This assumption does not hold in general if perspective projection is used and the viewing direction has a nonzero component in the direction of increasing function value. Figure 1 illustrates such a case. Wright's algorithm (ref.7) is also based on this assumption, and in addition is inexact because the perimeter is approximated by a vector of its heights at equally spaced points.

We will work in three-space with rectangular (x,y,z) coordinates. It will be assumed that the surface is being viewed from a viewpoint VP = (VP_x, VP_y, VP_z) with viewing direction VA = (VA_x, VA_y, VA_z). Object points are projected through VP onto an "image space" normal to VA. We require that the entire grid surface be contained in an open half-space whose boundary contains VP and has normal VA. A method for finding a viewing direction satisfying this requirement whenever one exists is given in ref. 1. An equivalent condition is that the image be bounded. We will be concerned with computing the entire image; of course, clipping in the image space may also be needed in practice.

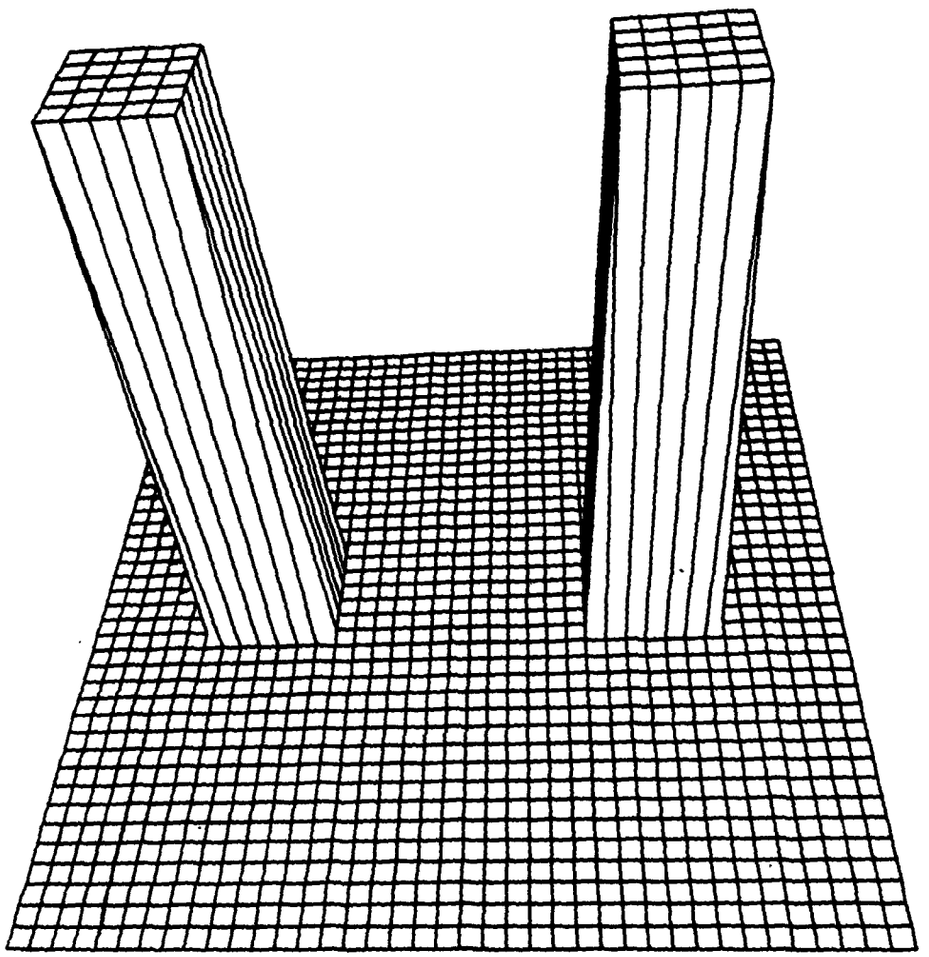


Figure 1: A surface which would not be drawn correctly by some previous algorithms.

2) DEFINITION OF GRID SURFACE

Suppose $x_0, \ldots, x_n$ and $y_0, \ldots, y_m$ are strictly increasing sequences of real numbers (the spacing need not be uniform). Points in three-space of the form $(x_i, y_j, 0)$ will be called "grid points". Planes of the form $x=x_i$ or $y=y_j$ will be called "grid planes". The "grid element" with index (i,j) is the rectangle in the xy plane bounded by the planes $x=x_i$, $x=x_{i+1}$, $y=y_j$ and $y=y_{j+1}$. The image of a grid element under a function will be called a "facet".

Suppose that the values of a bivariate function are given on a set of grid points. A "grid surface" is a continuous function g from R^2 to R which has the given values at the grid points and also satisfies: (1) g is linear between adjacent grid points and (2) the projection of any facet (i.e., image of a grid element under g) is bounded by the projection of the boundary of the facet. These conditions ensure that the image of the grid surface consists only of line segments which are projections of facet boundaries. The conditions are satisfied, for example, by the least-area surface containing the given points.

A surface defined over a rectangular region $[s_1,s_2]x[t_1,t_2]$ will be said to be viewed "face-on" if $VP_x \in [s_1,s_2]$ and $VP_y \in [t_1,t_2]$. It is viewed "edge-on" if exactly one of these containments holds, and "corner-on" if neither holds. These terms will be needed when discussing

the position of the viewpoint relative to facets or to the grid surface.

3) MOTIVATION FOR THE ALGORITHM

Consider the following method for solving the hidden line or hidden surface problem for an object consisting of facets or other polygons satisfying condition (2) above:

(1) Enumerate the facets as $F_1, \ldots, F_k$ in such a way that if F_i occludes F_j then $i < j$. Call such an order "occlusion-compatible".

(2) Initialize a description of the polygonal perimeter of the region in the image space drawn so far (referred to henceforth as the "perimeter"). This region is initially empty.

(3) For each i from 1 to k draw the part of facet F_i (for shaded-region drawings) or its boundary (for line drawings) which lies outside the current perimeter, and then update the perimeter to include the facet.

In general, there are problems with this approach. First, an occlusion-compatible order may be hard to compute. If there are intersecting or mutually occluding facets then such an order does not exist. Secondly, the task of merging arbitrary polygons (to update the perimeter) is fairly complex, since in general there can be holes, disjoint

pieces, and various pathological cases.

For grid surfaces, however, the method works, and is simplified in two ways:

(1) For a given viewpoint it is easy to enumerate the facets in an occlusion-compatible order.

(2) When the facets are processed in such an order, the region drawn at any stage of the process has a simple form: it is connected (i.e. in one piece) and is star-convex (explained below). As a consequence it is computationally simple to update the perimeter as new facets are processed.

4) GEOMETRIC PROPERTIES OF GRID SURFACES AND THEIR PROJECTIONS

In this section we discuss further the above two claims about grid surfaces. First we will give an occlusion-compatible enumeration of the facets. This enumeration depends on the relative viewpoint; there are three cases (see figure 2):

Case A) The surface is viewed face-on. Suppose VP lies directly above or below grid element $(i_\emptyset, j_\emptyset)$. The enumeration is given in the notation of FORTRAN implied do-loops. For future reference, the relative orientation of the facets is also given.

$(i_\emptyset, j_\emptyset)$	(face-on)
$((i_\emptyset, j), j=j_\emptyset+1, m-1)$	(edge-on)

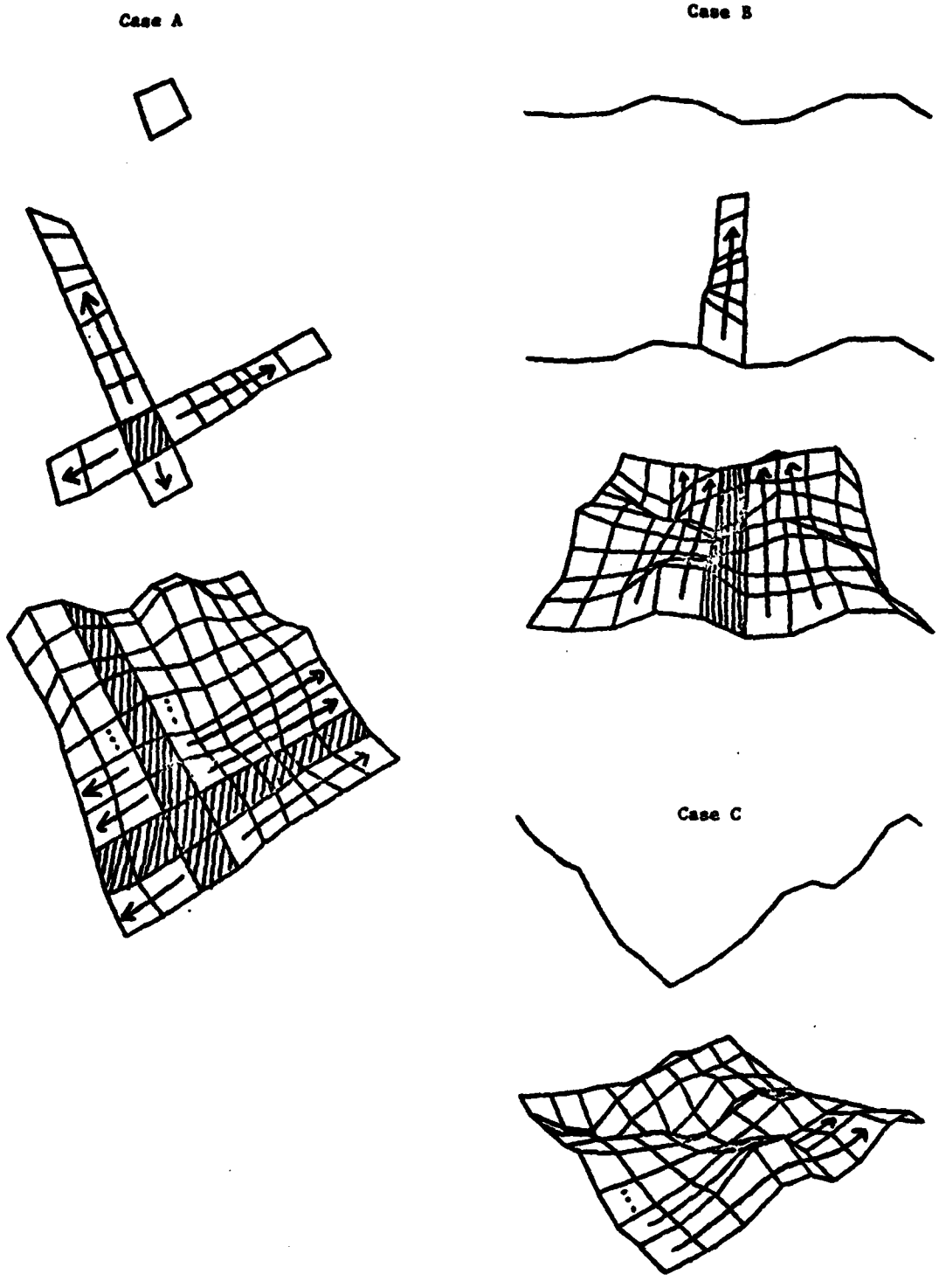


Figure 2: The initial perimeter and the facet processing order.

$((i_0,j),j=j_0-1,0,-1)$ (edge-on)

$((i,j_0),i=i_0+1,n-1)$ (edge-on)

$((i,j_0),i=i_0-1,0,-1)$ (edge-on)

$(((i,j),i=i_0+1,n-1),j=j_0+1,m-1)$ (corner-on)

$(((i,j),i=i_0+1,n-1),j=j_0-1,0,-1)$ (corner-on)

$(((i,j),i=i_0-1,1,-1),j=j_0+1,m-1)$ (corner-on)

$(((i,j),i=i_0-1,1,-1),j=j_0-1,1,-1)$ (corner-on)

The enumeration begins with the facet which is viewed face-on, goes outwards along the row and column of that facet, and finally fills in the four remaining rectangular areas.

Case B) The surface is viewed edge-on. Suppose $x_{i_0} \leq VP_x < x_{i_0+1}$ and $VP_y < y_0$. The enumeration is:

$((i_0,j),j=1,m-1)$ (edge-on)

$(((i,j),j=1,m-1),i=i_0+1,n-1)$ (corner-on)

$(((i,j),j=1,m-1),i=i_0-1,0,-1)$ (corner-on)

Case C) The surface is viewed corner-on, say $VP_x < x_0$ and $VP_y < y_0$. The enumeration is:

$(((i,j),i=1,n-1),j=1,m-1)$ (corner-on)

The other possibilities are equivalent to one of the above under symmetry.

We now turn to the form of the perimeter. The following analysis applies not only to the perimeter of the entire projected surface but also to the perimeter at any stage when the facets are processed in an occlusion-

compatible order.

The perimeter form depends on both the relative viewpoint and the orientation. There are three cases (see figure 3):

Case 1) (Overhead view) The surface is viewed face-on. Under the assumption that the image is bounded, the viewing direction must have a nonzero z component, so projected z-direction lines converge to a point VZ in the image space (this is the vanishing point property of perspective projection). VZ is the intersection of the viewing plane and the z-direction line passing through VP. In a polar coordinate system for the image space whose origin is VZ, the perimeter vertices have monotonic angles as the perimeter is traversed in one direction. If the viewpoint is not on a grid plane then the angles are strictly monotonic.

Case 2) (Oblique view) The surface is not viewed face-on and the viewing direction has a nonzero z component. Again, projected z-direction lines converge to a point VZ, and we consider a polar coordinate system with origin VZ. In this case the perimeter lies entirely within an acute sector with vertex VZ, and any ray in this sector intersects the perimeter exactly once or twice. Hence the perimeter has "inner" and "outer" parts, each of whose vertices, when traversed in one direction, have monotonic angles. The two parts share endpoints, and at any angle the outer part is at

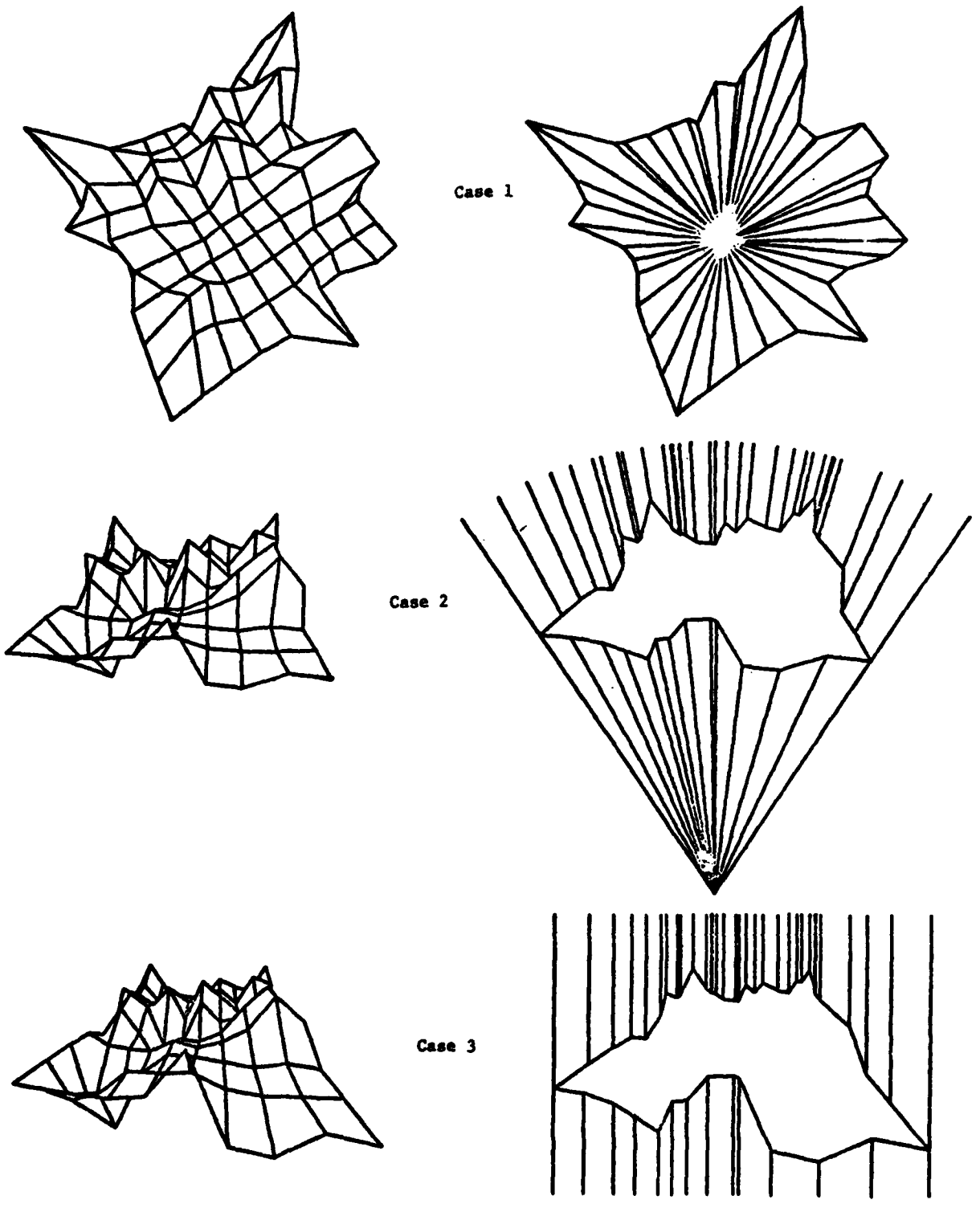


Figure 3: The forms of the perimeter.

least as far from VZ as is the inner part. If the viewpoint does not lie on a grid plane then the angles of the vertices are strictly monotonic.

In cases 1 and 2 the projection of the surface at any stage is star-convex around VZ. This means that if two image points lie on a ray from VZ, then all points on the segment between them are also in the image. In case 1, VZ is inside the perimeter and in case 2 it is outside.

Case 3) (Horizontal view) The viewing direction has zero z component. Here projected z-direction lines are parallel in the image space. If we call this direction up/down, then the perimeter has top and bottom parts each of whose vertices go monotonically left to right. If the viewpoint does not lie on a grid plane then the vertices are strictly monotonic left to right.

5) SOME COMPUTATIONAL DETAILS

As shown above, the natural coordinate system in which to represent the perimeter is either a polar coordinate system having as its origin the vanishing point of projected z-direction lines (overhead and oblique views) or a rectangular system in which one coordinate direction is that of projected z-direction lines (horizontal view). In what follows we will focus on the first two cases. The relevant parts of the analysis apply also to case 3 if "angle" and

"radius" are replaced by right/left and up/down coordinates relative to some fixed point.

We now discuss the polar coordinate system in more detail. First, if we wish to assign a unique angle to points in the image space there is necessarily a discontinuity in the angle function along the direction where the angle wraps around from 2π to 0. To exploit the angle monotonicity of the perimeter, it is necessary to define the angle function so that the discontinuity lies outside the portion of the perimeter currently being used. In case 2 this can be accomplished by making the discontinuity lie outside the sector containing the perimeter. In case 1 it is necessary to redefine the angle function in the middle of processing.

Secondly, the algorithm does not need to know the exact radius or angle of any point P, since only the angular and radial order of points are important. It suffices to compute functions r(P) and a(P) which are strictly increasing in the radius and angle respectively. Call such functions pseudo-angles and pseudo-radii respectively. The easiest pseudo-radius to compute is the square of the distance between P and VZ. The function

$$
\begin{aligned}
a(P) &= x/y && \text{if } -y \le x \le y \\
&= 2-(y/x) && \text{if } -x \le y \le x \\
&= 4+(x/y) && \text{if } y \le x \le -y \\
&= 6-(y/x) && \text{if } x \le y \le -x
\end{aligned}
$$

(where (x,y) = P - VZ) is a pseudo-angle discontinuous in the (-1,1) direction (see figure 4). By changing the additive constants in the four parts of the definition, the discontinuity can be made to be any of ($\pm$1,$\pm$1).

The main data structures used by the algorithm are the cartesian coordinates, pseudo-angles and pseudo-radii of the projected grid surface points and of the vertices of the two parts of the perimeter. These are stored in matrices for the grid points and doubly linked lists for the perimeter vertices.

Assume for the remainder of this section that the viewpoint does not lie in any grid plane, so that it is unambiguous whether the surface or a facet is viewed face, edge or corner-on.

It is convenient to initialize the perimeter not as empty but as consisting of the facet edges which are always visible given the relative viewpoint. These segments are drawn as part of the initialization. The initial perimeter depends on the position of VP relative to the surface as follows (see figure 2):

Face-on: The initial perimeter is the boundary of the facet which is viewed face-on. This facet is not processed further.

Edge-on: The initial perimeter is the edge of the surface which faces the viewpoint.

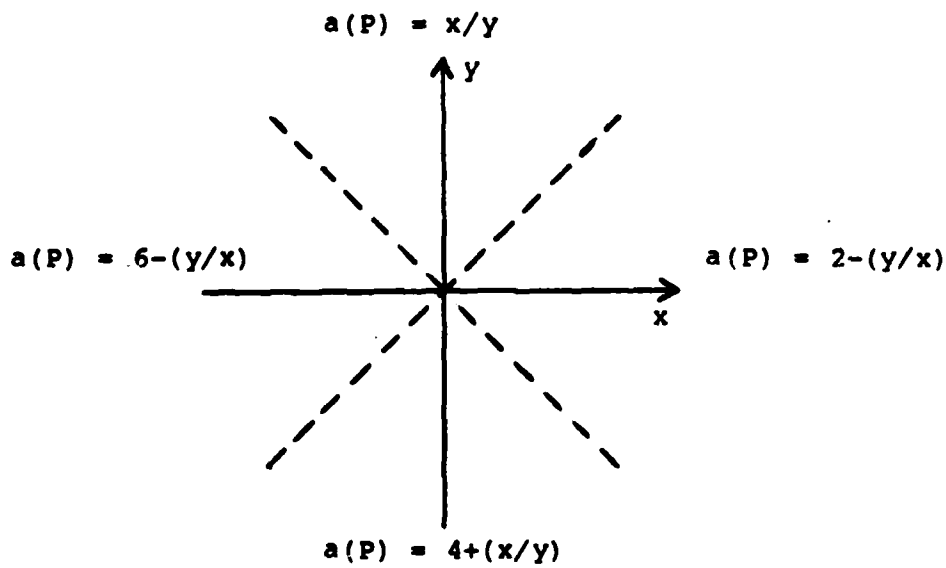


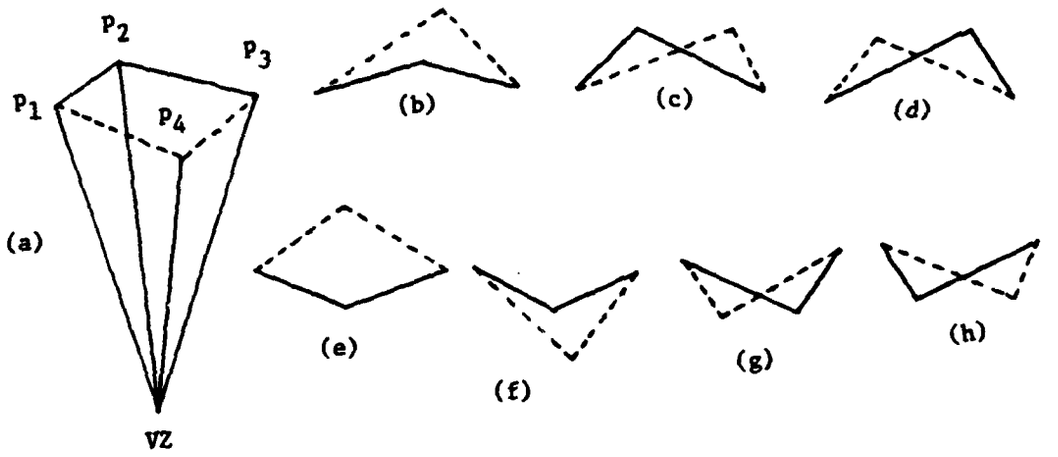
Figure 4: A pseudo-angle function discontinuous in the (-1,1) direction.

Corner-on: The initial perimeter is the two edges of the surface containing the corner which faces the viewpoint.

There are several benefits from initializing the perimeter in this way. One is that the angular extrema of the final perimeter are already attained in the initial perimeter, so we never have to worry about searching off the ends of the perimeter. More importantly, the perimeter initialization together with any occlusion-compatible facet processing order ensure that all facets can be dealt with according to one of the following two cases:

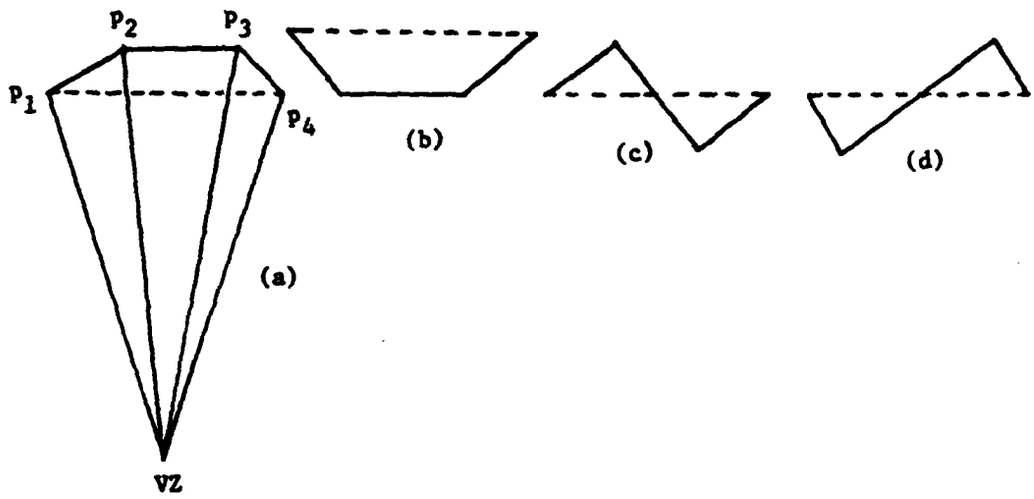
(1) If the facet is viewed edge-on then the facing edge will have already been processed by the time the facet is processed. There are four subcases for the relative positions of the edges of the facet. In each subcase it is possible to deduce that some or all of the facet edges cannot leave the inner or outer perimeters. The subcases are distinguished by the signs of the scalar products $((P_2-P_1)^r,(P_3-P_1))$, $((P_2-P_1)^r,(P_4-P_1))$, and $((P_3-P_2)^r,(P_4-P_2))$, where $(x,y)^r$ denotes $(y,-x)$ and $P_1,\ldots,P_4$ are the projected facet vertices in order of increasing angle (see figure 5).

(2) If the facet is viewed corner-on then the two facing edges will have already been processed by the time the facet is processed. The are six subcases for the relative positions of the edges (see figure 6).



	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
$((p_2-p_1)^r,(p_3-p_1))$	+	+	+	+	-	-	-	-
$((p_2-p_1)^r,(p_4-p_1))$	-	+	-	+	+	-	-	+
$((p_3-p_2)^r,(p_4-p_2))$	-	+	+	-	+	-	+	-
vertex indices of part which may leave outer perimeter	123	-	123	123	-	123	12	23
vertex indices of part which may leave inner perimeter	-	123	23	12	123	-	123	123

Figure 5: The eight classes of facets viewed corner-on. Dotted lines represent the closer edges, which have already been processed.



	(a)	(b)	(c)	(d)
$((p_4-p_1)^r,(p_2-p_1))$	-	+	-	+
$((p_4-p_1)^r,(p_3-p_1))$	-	+	+	-
vertex indices of part which may leave outer perimeter	1234	-	123	234
vertex indices of part which may leave inner perimeter	-	1234	234	123

Figure 6: The four classes of facets viewed edge-on.

Thus we are left with the following simplified problem: given a perimeter part (say the outer part) whose vertices have strictly increasing angles, and a list of two, three, or four points in strictly increasing angular order such that the first and last are not outside the perimeter, find the visible parts of the new segments and update the perimeter to include them. A fast way to do this is to scan concurrently through the facet and perimeter vertex lists in order of increasing angle, searching for points where the facet boundary leaves or enters the perimeter. When a matched exiting and entering crossing pair is found, the perimeter vertices between the two points are deleted from the perimeter, and the facet boundary vertices between the two points are added. The line segments between the crossings and the intervening facet vertices are drawn (see figure 7). In the case of shaded drawings, the region bounded by these segments and the deleted portion of the perimeter is drawn.

It is not necessary to search the entire perimeter for the first point in the angular range of the current facet. This is because the facet processing order given above goes along rows and columns. By maintaining pointers into the perimeter near the most recent facet and near the first facet of the current row or column, global searching can be avoided.

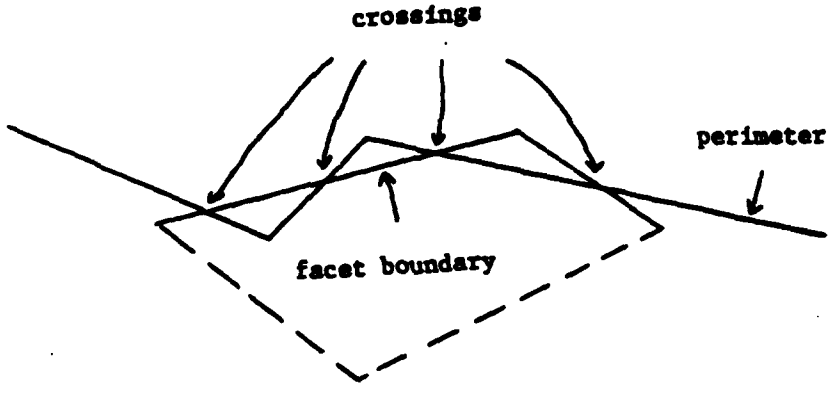


Figure 7: An example of appending a facet to the perimeter.

Note that the crossing pairs will occur sequentially in the search through the lists. The search ends when the last facet vertex is reached, so only the perimeter vertices within the angular range of the facet are scanned. The task of deciding when and where the crossings occur can make use of the pseudo-distances of the various vertices. The only types of arithmetic calculations that are necessary are to decide if a point is to the right or left of a line (this need be decided at most once for each vertex in the search) and to calculate the intersection of a pair of segments (this need be calculated only for crossings which are the proper intersection of a perimeter and a facet segment).

When the surface is viewed face-on there is no inner perimeter. In the other cases the perimeter has inner and outer parts, and the visible portion of a facet is that which is inside the inner part or outside the outer part. It is desirable to have the same routine handle both parts. This can by done by negating the angles and radii of the vertices of the inner perimeter and the facet vertices, and then proceeding as for the outer perimeter.

6) SPECIAL CASES OF THE RELATIVE POSITION

We now consider the cases where VP lies in a grid plane, say $VP_x = x_i$. A difficulty arises because the perimeter will no longer have strictly monotonic angles. This possibility makes the facet processing logic considerably more complicated. The problem can be circumvented, however, by processing separately the parts of the grid on different sides of the plane $x = x_i$. This separation is possible because neither part can ever occlude the other. Each part is processed as described in the previous section, except that one or both angular extrema (depending on whether the surface is viewed edge-on or face-on) must be treated as a special case.

If the viewpoint lies above a grid point not on the boundary of the grid (say $(VP_x, VP_y) = (x_i, y_j)$) then the surface must be divided into the four parts determined by the planes $x = x_i$ and $y = y_j$. The four parts are mutually non-occluding and can be drawn as described above, with both ends of the perimeter treated as special cases (see figure 8).

In many applications it is not necessary to implement special handling of the above cases. Instead the program can move the viewpoint slightly if it lies in a grid plane. For applications involving animation, where the viewpoint might move continuously through grid planes, it would be worthwhile to add the extra logic. A similar remark applies

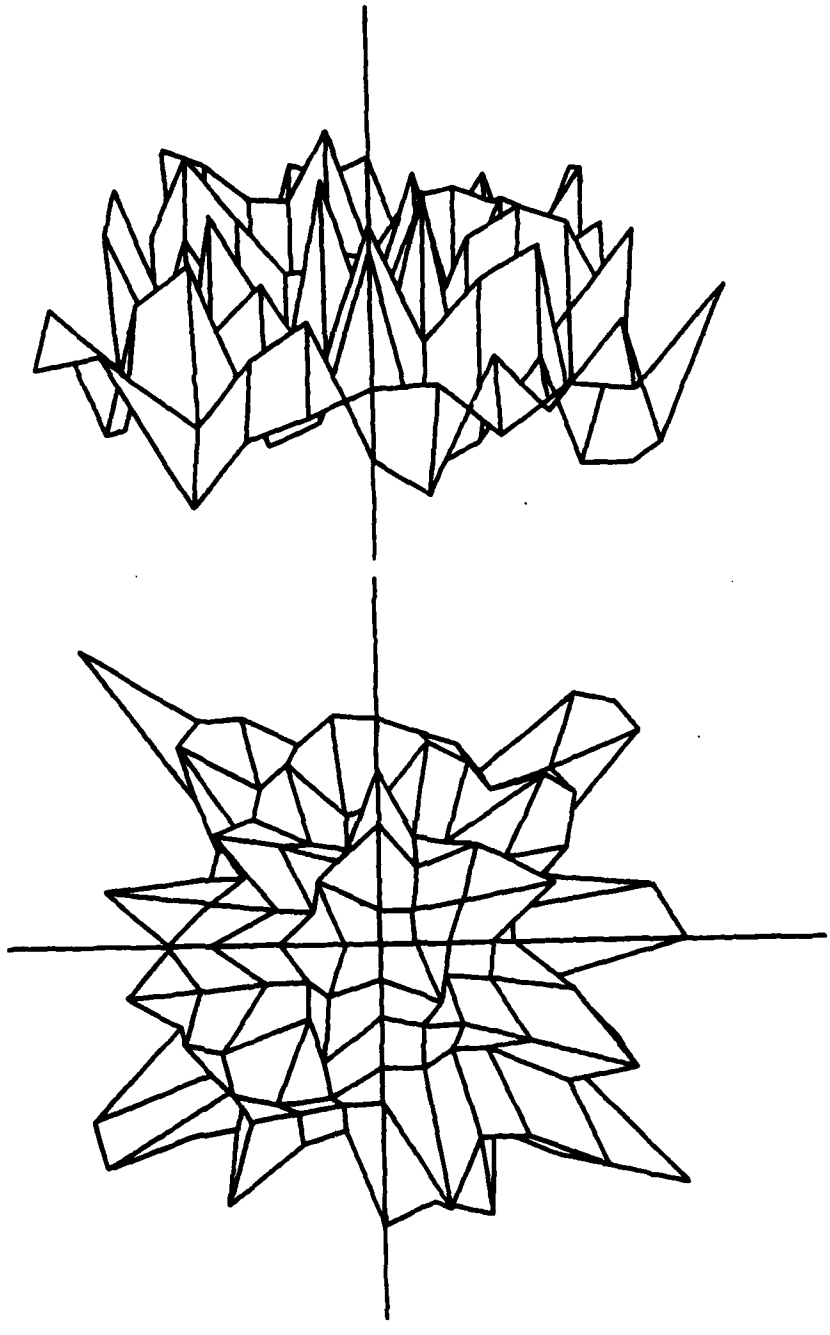


Figure 8: Cases where the viewpoint lies in a grid plane.

to case 3 (horizontal view), which can be changed to case 2 by an arbitrarily small perturbation of VA.

7) DRAWING OTHER OBJECTS

A program for drawing grid surfaces should allow the user to draw projected objects other than the surface itself. Hidden line elimination for the additional objects can be done within the framework of the present algorithm as long as the objects are of the wire-frame type and each object either does not occlude the grid surface or is not occluded by it. In the latter case the object is drawn in its entirety. In the first case only those portions of the object which lie outside the perimeter are drawn. For a given line segment in the object, this visible portion consists of the parts which lie outside the angular range of the perimeter, lie further than the outer perimeter, or lie closer than the inner perimeter. These parts are easily calculated, the latter two using the same methods as for drawing the facets.

The author's implementation allows axis grids, text, and contour maps to be drawn. All of these can be positioned and oriented arbitrarily in three-space and are drawn in perspective. An example showing these features is given in figure 9.

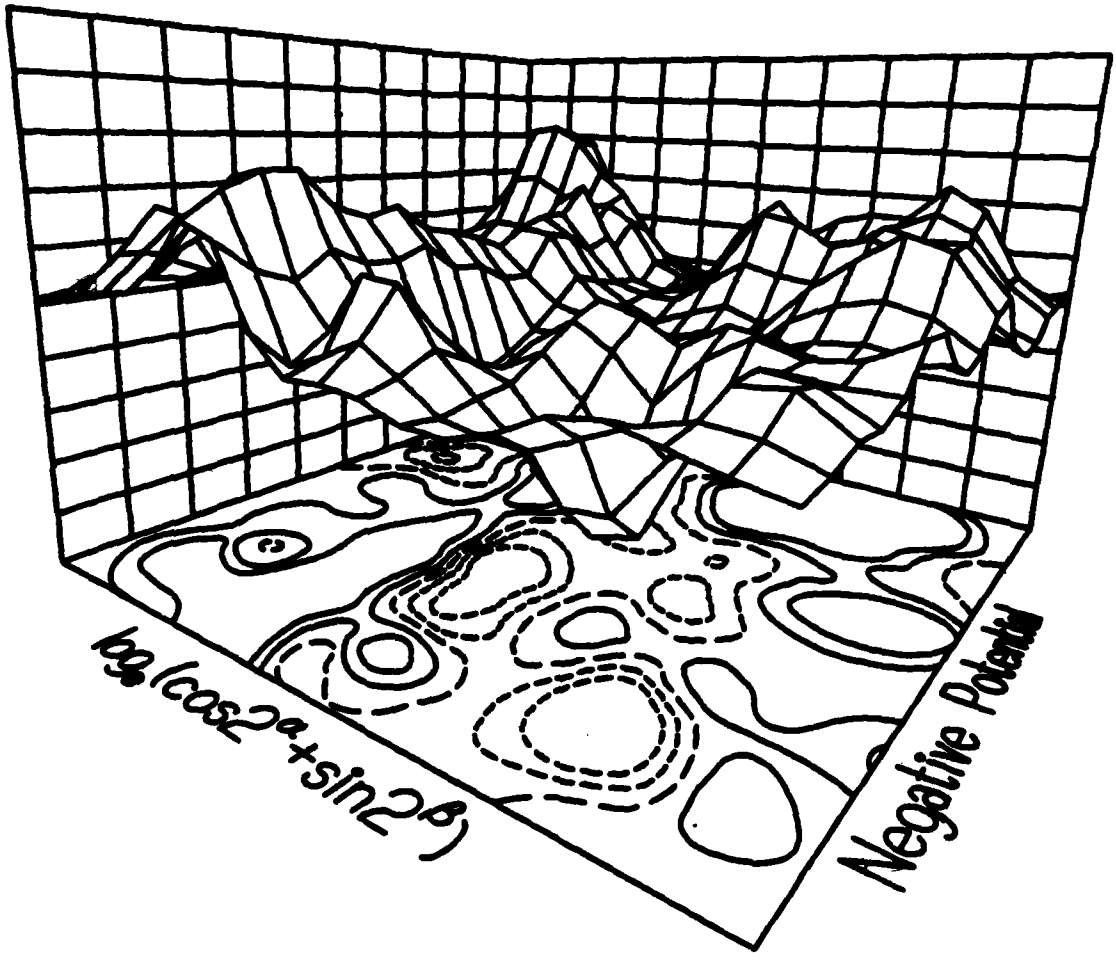


Figure 9: A surface image with secondary objects (axis grids, text and contour map).

8) PERFORMANCE

The methods described in this paper were implemented in a FORTRAN program on a Harris Slash 7 minicomputer with floating point hardware. The program has been tested on many surfaces. Performance results are given in Table 1 for the function $f(x,y) = \exp(-x^2-y^2)$ and for a surface consisting of samples from the uniform random distribution on [0,1]. These were generated on uniform grids of varying fineness on the square [-2,2] x [-2,2]. The resulting surfaces were drawn from the viewpoint (6,8,3) (see figure 10). It can be seen from the graph in figure 11 that execution time, in these cases, appears to increase approximately linearly as a function of the number of facets.

These results suggest that the algorithm as implemented is approximately linear-time for commonly-encountered surfaces. It is particularly interesting that the random surface, which appears to be at least as complex as any surface arising in applications, is handled in only about 50% more time than the smooth surface.

A proof of linear expected or worst-case time would require an upper bound on the average number of perimeter vertices in the angular range of each facet. Finding such a bound (possibly for a restricted class of surfaces) is posed as a challenge to interested readers.

TABLE 1

Grid size	Number of facets	Execution time (seconds)	
		Smooth surface	Random surface
10 by 10	81	.184	.221
20 by 20	361	.725	.921
30 by 30	841	1.623	2.315
40 by 40	1521	2.886	3.969
50 by 50	2401	4.527	6.515
60 by 60	3481	6.532	9.534

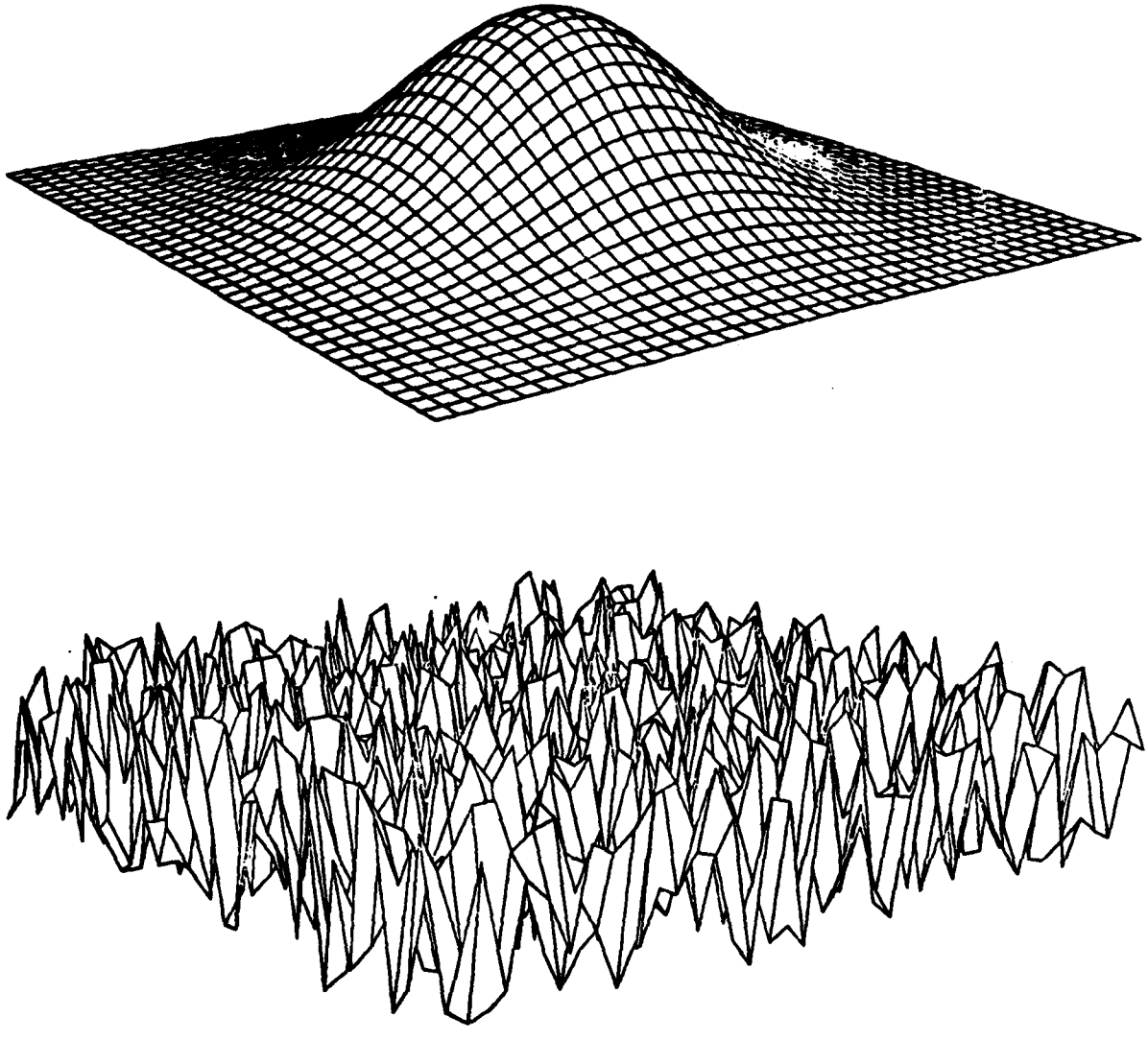


Figure 10: Smooth and random surfaces used as test cases.

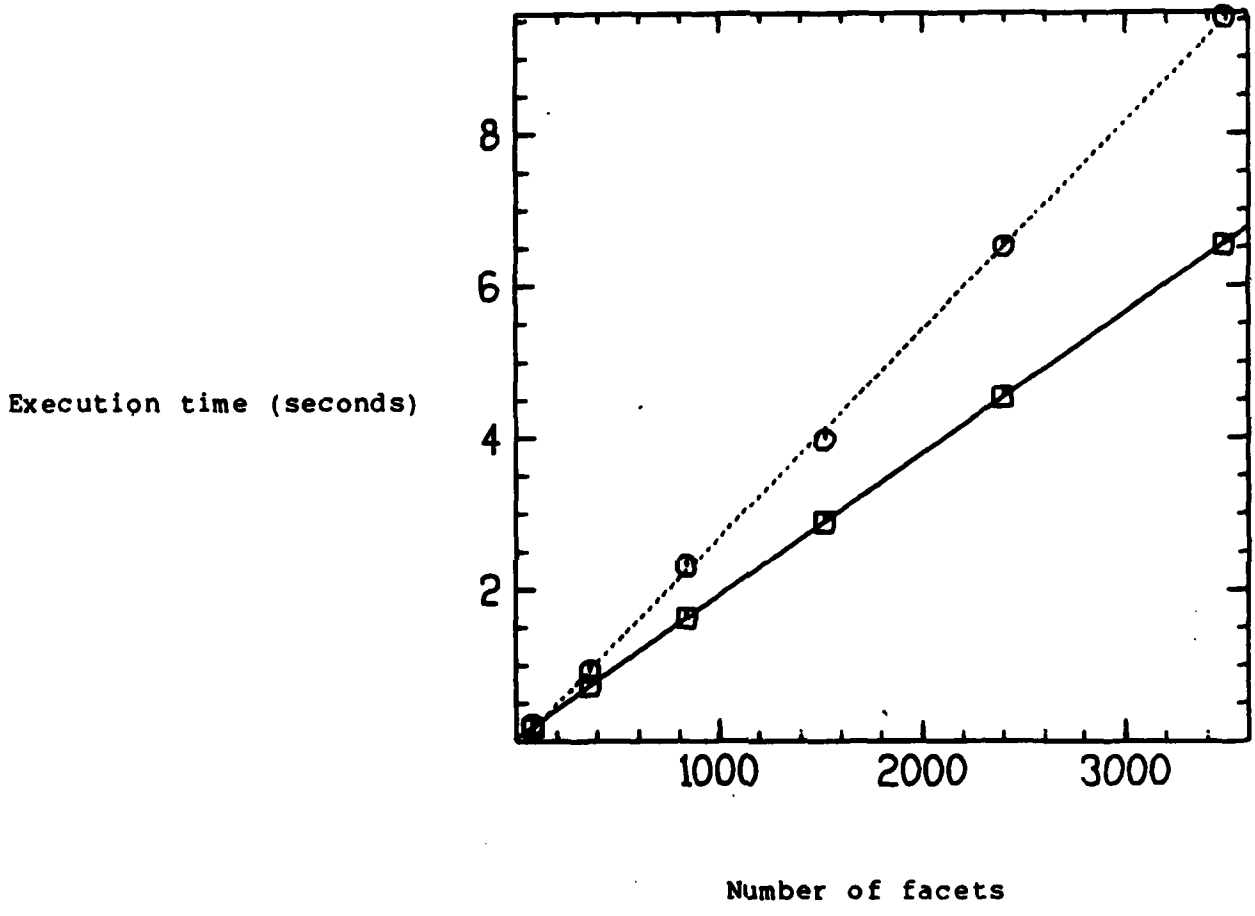


Figure 11: Execution time data with best-fit lines for smooth (solid line) and random (dotted line) surfaces.

9) CONCLUSION

It has been shown that projected images of grid surfaces have a geometric structure which allows a fast and fairly simple algorithm for hidden line and hidden surface elimination. The key properties are:

1) The existence of a trivially computable occlusion-compatible order for processing the facets.

2) The star-convexity of the perimeter, which makes it easy to represent and modify the perimeter and also means that only the perimeter segments in a small angular neighborhood of each facet need be checked for crossings.

The implementation of the algorithm has proved to be very robust and efficient. It is possible that some of the ideas behind the algorithm may be relevant to hidden line and hidden surface elimination for other classes of objects.

REFERENCES

(1) Anderson, D.P. An orientation method for central projection programs. Computers and Graphics 6,1 (1982), 35-37.

(2) Butland, J. Surface drawing made simple. Computer-aided Design 11,1 (January 1979), 19-22.

(3) Franklin, W.R. A linear-time exact hidden surface algorithm. Computer Graphics 14,3 (July 1980), 117-123.

(4) Kubert, B.R., Szabo, J. and Gulieri, S. The perspective representation of functions of two variables. JACM 15,2 (April 1968), 193-204.

(5) Sutherland, I.E., Sproull, R.F. and Shumacker, R.A. A characterisation of ten hidden surface algorithms. Computing Surveys 6 (1974), 1-55.

(6) Williamson, H. Algorithm 420 - hidden-line plotting program (J6). CACM 15,2 (February 1972), 100-103.

(7) Wright, T.J. A two-space solution to the hidden line problem for plotting functions of two variables. IEEE Trans. Comput. c22,1 (January 1973), 28-33.

SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: 2447	2. GOVT ACCESSION NO.: AD-A124348	3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): HIDDEN LINE ELIMINATION IN PROJECTED GRID SURFACES		5. TYPE OF REPORT & PERIOD COVERED: Summary Report - no specific reporting period
		6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): David P. Anderson		8. CONTRACT OR GRANT NUMBER(s): DAAG29-80-C-0041 MCS-8200632
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Mathematics Research Center, University of Wisconsin 610 Walnut Street Madison, Wisconsin 53706		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Work Unit Number 5 - Mathematical Programming
11. CONTROLLING OFFICE NAME AND ADDRESS: See Item 18 below.		12. REPORT DATE: December 1982
		13. NUMBER OF PAGES: 30
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report): UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

National Science Foundation
Washington, D. C. 20550

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

hidden line elimination
hidden surface elimination
function graphing
grid surface

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Hidden line and hidden surface problems are often simpler when restricted to special classes of objects. An example is the class of grid surfaces, i.e. graphs of bivariate functions represented by their values on a set of grid points. Projected grid surfaces have geometric properties which permit hidden line or hidden surface elimination to be done more easily then in the general case. These properties are discussed in this paper and an algorithm is given which exploits them.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)


DATE:
ILME